Transitioning to Mac OS X

An Overview of the **Carbon** Programming Interface

Apple Computer, Inc.

Transitioning to Mac OS X

An Overview of the **Carbon** Programming Interface

5/9/98

Contents

C H A P T E R 1

Introducing Carbon

Contents

Introducing Carbon

This chapter introduces Carbon and describes the key benefits it offers you, the Mac OS developer.

Carbon represents the core set of programming interfaces you can use to build Mac OS X applications that can also be deployed on Mac OS 8. Using Carbon you can take advantage of preemptive multitasking, memory protection, and dynamic resource allocation in Mac OS X while maintaining maximum source code compatibility with your current applications.

Carbon includes most of the Mac OS functions that developers rely on today, while adding a small number of new functions to support Apple's goal of providing a more robust, responsive, and productive computing experience for our customers.

The purpose of this document is threefold:

- Introduce Carbon and describe the features and benefits it offers developers.
- Help you assess the degree to which your existing applications are compatible with Carbon.
- Enlist your assistance in prioritizing Apple's development efforts.

This document is a preview of Apple's plans for Mac OS X and Carbon. It is not intended to be the definitive guide. We're providing early information about Carbon so you have the opportunity to evaluate how well it meets your development needs. Your input will play an important role as we define and implement the Carbon programming interface.

The Carbon Advantage

The Mac OS has proven to be the strongest development platform for building innovative applications. Thousands of world-class programs have been created on the Macintosh, and the Mac OS continues to be the platform of choice for creative professionals in the design, publishing, education, and new media markets. Flexible, extensible, and complete, the Mac OS has matured and evolved while retaining its leading-edge characteristics.

Mac OS X brings important new features and enhancements that developers have asked for—without forcing you to adopt an entirely new API. Because Carbon supports most of the functions you're using today, you'll be able to easily integrate these new capabilities.

When running in Mac OS X, Carbon applications gain these advantages:

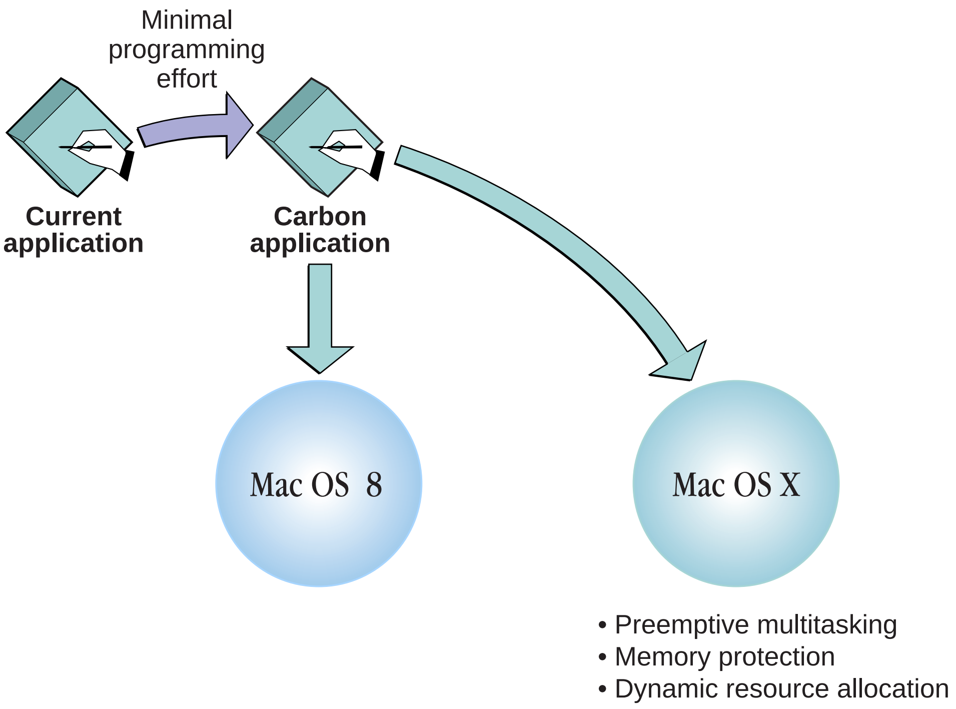
- **Greater stability**
  Preemptive multitasking and protected address spaces will help prevent errant applications from crashing the system or other applications.
- **Improved responsiveness**
  Each application will be guaranteed processing time through preemptive scheduling, resulting in a more responsive user experience.
- **Efficient use of system resources**
  Applications will dynamically use memory and other system resources based on actual needs rather than predetermined values.

Apple's overriding goals for Carbon are simple:

- **Maximum source code compatibility with existing applications**
  Apple is committed to making the transition to Mac OS X as easy as possible for you by preserving your investment in Mac OS source code.
- **Carbon applications can run on Mac OS 8**
  You will not need to maintain separate source code versions because Carbon supports both the Mac OS 8 and Mac OS X runtime environments. As always, you should use the Gestalt Manager to test for the existence of specific features before using them.

Carbon lets you take advantage of the many benefits of Mac OS X while leveraging source code written to the current API. As Apple moves the Mac OS forward, Carbon ensures you won't be left behind.

Figure 1-1 The Carbon advantage


Minimal programming effort
Current application
Carbon application
Mac OS 8
Mac OS X
• Preemptive multitasking
• Memory protection
• Dynamic resource allocation

A Pragmatic Approach

Mac OS X and Carbon are a direct response to the needs and concerns voiced by the Apple developer community. After taking a fresh look at previous modernization efforts, Apple has chosen a path that is modest in scope but will yield large gains for developers and users alike. This strategy preserves our mutual investment in the Mac OS platform and is designed to maximize the return on your development efforts.

Apple's goal is to ease your transition to Mac OS X by making as few changes as possible to the Mac OS API. Carbon accomplishes this goal by providing a compatible set of interfaces on which to base both existing and future applications.

By initially concentrating on the most commonly used set of Mac OS functions, Apple plans to deliver Carbon in a timely fashion so that you can begin adapting your applications as soon as possible. We expect the level of programming effort required for Carbon compatibility will be about the same as was needed for converting 68K applications to PowerPC.

Working Together

Apple recognizes that the success of Mac OS X depends on your willingness to adopt Carbon. While our efforts are focused on making this as easy as possible, we need your feedback to assist us in determining the Mac OS APIs that are most essential to your work.

To help you assess how much effort may be involved in adopting Carbon, Apple has developed an analysis tool that can examine your executable files and issue a report that will help you determine the changes you may need to make for Carbon compatibility. Chapter 3, “Preparing for Mac OS X,” describes this tool and the steps you can begin taking to simplify your transition to Carbon.

Transition Goals

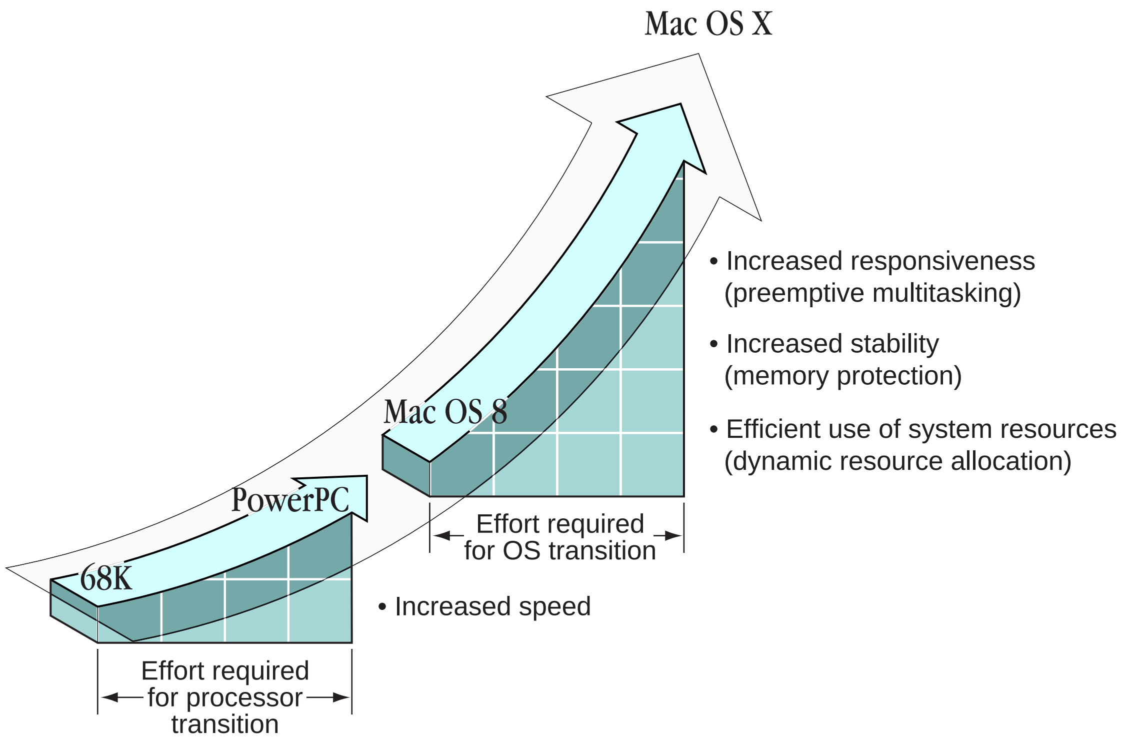
The successful transition from 68K to PowerPC is Apple’s model for the transition to Mac OS X. The projects are similar in that the adoption of the PowerPC processor involved significant changes to the basic foundation of the operating system, yet required relatively few changes to the Mac OS API.

One reason for the success of the PowerPC transition was that developers were able to quickly revise their applications. The resulting performance gains were impressive—creating a lucrative market for native applications and upgrades of existing products. Customer demand for Mac OS X applications will present similar business opportunities for developers.

Equally important, adopting Carbon allows you to develop for Mac OS X without impacting development of products that run on today’s installed base.

Figure 1-2 compares the Mac OS X transition to the PowerPC transition and illustrates how a modest effort can translate into sizable benefits.

Figure 1-2 The Mac OS X transition: a modest effort for dramatic gains


Mac OS X
Mac OS 8
PowerPC
68K
• Increased responsiveness (preemptive multitasking)
• Increased stability (memory protection)
• Efficient use of system resources (dynamic resource allocation)
Effort required for OS transition
• Increased speed
Effort required for processor transition

There are over twelve thousand commercial applications and countless noncommercial applications currently shipping for the Mac OS. Carbon validates and preserves this huge investment by minimizing the effort required to carry these applications forward.

Design Goals

Apple’s engineering efforts are focused on the following design goals. As work progresses we encourage you to let us know how well you think we’re meeting these goals.

- Carbon will maintain a high degree of source code compatibility with existing applications.
- Carbon applications can run on both Mac OS 8 and Mac OS X.
- Carbon will support preemptive multitasking, memory protection, and dynamic resource allocation in Mac OS X.
- Carbon applications can be built using existing development tools.

Send your comments and suggestions about Carbon to the development team using this e-mail address:

carbon@apple.com

C H A P T E R 2

Support for Existing Mac OS Services

Contents

The programming interfaces to the vast majority of Mac OS services remain unchanged in Carbon. A number of interfaces are supported with slight changes. Of the unsupported interfaces, most are for services that are either obsolete or have had limited developer or customer adoption. Other interfaces are restricted: to facilitate a more efficient and robust I/O system in Mac OS X, Carbon doesn't support application use of several low-level hardware interfaces. However, many of these existing interfaces can be called from driver-level software.

This chapter describes the levels of support that Carbon offers for the current Mac OS programming interface. Chapter 3, "Preparing for Mac OS X," describes a compatibility analysis tool that reports exactly which functions your application may be calling that Carbon doesn't support. The report generated by this tool also suggests how you can modify your application for Carbon.

Carbon is a work in progress, and Apple welcomes your comments and suggestions about our efforts. The feedback you send to carbon@apple.com will help us deliver Carbon in a manner that most quickly advances your software as well as the Mac OS platform.

This chapter begins with a discussion of how Carbon and Mac OS X relate to the Mac OS 8 application model. The rest of the chapter summarizes Carbon support for the existing Mac OS programming interface.

Carbon and the Mac OS Application Model

The Mac OS 8 application model remains fundamentally unchanged in Mac OS X. Your application will employ system services in essentially the same manner for both operating systems. Applications on both systems, for example, will be built around the event loop and the `WaitNextEvent` function; they'll call the Human Interface Toolbox to implement standard elements of the user interface; they'll use the Gestalt Manager to test for system features; and they'll use such services as the Memory Manager and the Process Manager to acquire low-level operating system resources.

But because Mac OS 8 and Mac OS X are built on different architectures, there will be slight differences in the way your application uses some system services. This section highlights aspects of the application model that are most important to developers and explains where those aspects might diverge across operating systems.

Preemptive Scheduling and Application Threading

In Mac OS X, each Carbon application is scheduled preemptively against other Carbon applications. For calls to most low-level operating system services, Mac OS X will also support preemptive threading within an application. Because most Human Interface Toolbox functions are not reentrant, however, a multithreaded application will initially be able to call these functions only from cooperatively scheduled threads. In both Mac OS 8 and Mac OS X, an application can use the Thread Manager to create cooperatively scheduled threads.

Thread-based preemptive access to all system services—including the Human Interface Toolbox—is an important future direction for the Mac OS. Until all system services become fully reentrant, Apple will specify which Carbon routines are safe for preemptive threads in Mac OS X.

Apple will also provide functions for specifying your application's scheduling requirements in Mac OS X.

Separate Application Address Spaces

In Mac OS X, each Carbon application will run in a separate address space. An application can't reference memory locations—or corrupt another application's data—outside of its own address space. This separation of address spaces increases the reliability of the user's system, but it may require small programming changes to applications that use zones, system memory, or temporary memory. For example, temporary memory allocations in Mac OS X will be allocated in the application's address space, and Apple will define new functions for sharing memory between applications. See "Memory Manager" (page 62) for information about memory management for Carbon applications.

Virtual Memory

Mac OS X will employ a dynamic and highly efficient virtual memory system. This virtual memory system will introduce a number of changes in the addressing model. For example, Carbon applications should assume virtual memory is always on, and Apple will provide functions for specifying application memory-swapping requirements in Mac OS X. As described in "Memory Manager" (page 62), developers who haven't strictly followed Apple guidelines in their use of virtual memory may need to revise their applications for Carbon.

Code Fragments and the Code Fragment Manager

Carbon fully supports the programming interface for the Code Fragment Manager, and the Mac OS X runtime environment fully supports code compiled into code fragments. For Mac OS X, however, all code fragments must contain PowerPC code.

Mixed Mode Manager

For source code compatibility, Carbon supports universal procedure pointers (UPPs) transparently. Because Mac OS X will not run 68K code, the Mixed Mode Manager will not provide any useful functionality on that operating system. However, you may keep Mixed Mode Manager calls in your application to maintain source code compatibility with Mac OS 8.

Standard and Custom Definition Procedures

Carbon supports the standard Mac OS 8 definition procedures (also known as **defprocs**) for such human interface elements as windows, menus, and controls. Custom definition procedures are also supported, as long as they are compiled as PowerPC code.

Callback Functions

Carbon supports most Mac OS callback functions. Mac OS X will fully support callback functions within an application's address space. In Carbon, callback functions have native PowerPC conventions instead of the previous 68K conventions, but Carbon doesn't change these function definitions.

The Trap Table

Because the purpose of the trap table is to provide entry points to the Mac OS for 68K code, the trap table will be irrelevant in Mac OS X and other future

PowerPC-native versions of the Mac OS. To help ensure that it complies with Carbon, your application should not dispatch calls through the trap table. Furthermore, your application should not patch any traps. If your application relies on any patches, please tell us why so that we can help you remove this dependency.

Control Panels

To support systemwide appearance, control panels in Mac OS 8 are being implemented as applications instead of components. For both Mac OS 8 and Mac OS X, developers should reimplement their control panels as applications.

Data Structure Access

So that future versions of Mac OS X can support access to all system services through preemptive threads, Carbon begins to limit direct application access to data structures used by the Mac OS. Carbon allows one of four types of access to a data structure, depending on which access is appropriate for that structure:

- Your application can read from and write to a data structure without restriction.
- Your application can read from and write to a data structure, but after modifying the structure, your application must call a function alerting the operating system that the structure has been changed.
- Your application can read from a data structure, but your application can modify the structure only by using an accessor function.
- Your application has no direct access to a data structure. Instead, your application can obtain and set values in the structure only by using accessor functions.

The rest of this chapter describes general levels of support Carbon offers for the current Mac OS programming interface. For this discussion, the programming interface is divided into related technologies. For more complete information about any particular portion of the interface, see Chapter 4, “Carbon Reference.”

Human Interface Toolbox

The Human Interface Toolbox consists of a group of libraries that implement the Mac OS human interface. Nearly all of the programming interfaces to these libraries are supported in Carbon. The major exception is the Standard File Package, which is not supported. In place of the Standard File Package, you are encouraged to begin adopting Navigation Services in your current Mac OS development efforts. (To assist in the switch to Navigation Services, Apple is making this technology compatible back to System 7.5.5.)

Applications that use the standard Mac OS 8 definition procedures (also known as **defprocs**) for such human interface elements as windows, menus, and controls inherit the current Mac OS 8 human interface appearance. Applications that use custom definition procedures work correctly, but because custom definition procedures invoke their own drawing routines, Mac OS 8 can't draw these applications with the current appearance. You are encouraged to adopt the standard Mac OS 8 definition procedures in your applications. Furthermore, to be compliant with Carbon, custom definition procedures can no longer contain 68K code. If you must use custom defprocs, make sure they are compiled as PowerPC code.

The Human Interface Toolbox in Mac OS 8 defines data structures that are used extensively by applications. These data structures include menu, control, and window records. The Human Interface Toolbox also defines functions for creating and disposing of these structures. These functions include `GetMenu`, `DisposeMenu`, `GetNewCWindow`, and `DisposeWindow`. Apple encourages you to use only the functions defined by the Human Interface Toolbox for creating and disposing of its data structures. Apple further recommends that your application gain access to these structures only through accessor functions. Using these functions, soon to be provided, and limiting your application's access to Toolbox data structures will help make your application fully thread safe in the future.

For more information on the individual technologies that make up the Human Interface Toolbox, please see the following:

Appearance Manager (page 46)	List Manager (page 59)
Control Manager (page 50)	Menu Manager (page 64)
Dialog Manager (page 52)	Navigation Services (page 66)

Finder Interface (page 56)	Picture Utilities (page 69)
Help Manager (page 58)	Standard File Package (page 76)
Icon Utilities (page 58)	Window Manager (page 80)

Application Utilities

The application utilities consist of facilities for extending human interface features and data-sharing capabilities beyond those provided by the Human Interface Toolbox. Carbon supports nearly all of the programming interfaces to these utilities. The major exceptions are the Edition Manager, which is only partially supported, and AV components and the Data Access Manager, which are not supported. The programming interface defined by the Speech Manager is currently under review. (We welcome your feedback on the importance of the Speech Manager in your development efforts.)

To support the Edition Manager, which has had limited customer adoption, Carbon will provide a mechanism for your application to open and read files that contain editions.

Carbon does not support the Data Access Manager. The Data Access Manager has become an outmoded interface for applications that communicate with databases and is largely unused by developers.

The AV components programming interface allowed developers to extend the Monitors and Sound control panel. Apple will define a new programming interface to provide similar functionality in Carbon.

For more information on the individual technologies that make up the application utilities, please see the following:

Apple Guide Manager (page 46)	Folder Manager (page 56)
AV Components (page 47)	Location Manager (page 60)
Control Strip Services (page 51)	Open Scripting Architecture (page 66)
Data Access Manager (page 51)	Scrap Manager (page 73)

Display Manager (page 54)	Speech Manager (page 76)
Drag Manager (page 54)	Translation Manager (page 79)
Edition Manager (page 54)	HyperCard XCommand Support (page 58)

Graphics Services

The graphics services allow your application to construct and render images. Carbon supports QuickDraw, the Printing Manager, QuickDraw 3D, the ColorSync Manager, the Color Picker Manager, and the Palette Manager. However, QuickDraw GX is not supported in Carbon. QuickDraw GX has had limited adoption by developers and customers; in its place, you should use other Mac OS imaging and typography managers, such as QuickDraw and the Font Manager.

Apple is currently reviewing the extent to which Carbon will support Game Sprockets. Please give us feedback on the importance of Game Sprockets in your development efforts.

For more information on the individual technologies that make up the graphics services, please see the following:

Color Picker Manager (page 48)	Printing Manager (page 70)
ColorSync Manager (page 49)	QuickDraw (page 71)
Game Sprockets (page 57)	QuickDraw 3D (page 72)
Palette Manager (page 68)	QuickDraw GX (page 72)

Multimedia Services

The multimedia services assist your application in combining multiple forms of communication, such as text, pictures, video, sounds, and music. Carbon supports nearly all of the programming interfaces for these services. The exception is the MIDI Manager, for which Apple has already discontinued support.

Note
Your application can, of course, employ Apple's QuickTime collection of cross-platform multimedia technologies on both Mac OS 8 and Mac OS X. ◆

For more information on the individual technologies that make up the multimedia services, please see the following:

Image Compression Manager (page 59)
MIDI Manager (page 64)
Movie Toolbox (page 65)
QuickTime Components (page 73)
Sound Manager (page 75)

Text and Other International Services

Apple supplies programming interfaces for text and other international services to help you create applications that are adaptable for worldwide markets. Carbon fully supports the programming interfaces to all of these services, with the exception of the Dictionary Manager. (The Dictionary Manager was intended to help applications handle text conversion for 2-byte script systems.) Apple will provide a Carbon replacement for the Dictionary Manager.

For more information on the individual technologies that make up text and other international services, please see the following:

Apple Type Solution for Unicode Imaging (page 47)
Dictionary Manager (page 53)
Font Manager (page 57)
International Resources (page 59)
QuickDraw Text (page 72)
Date, Time, and Measurement Utilities (page 52)
Text Encoding Conversion Manager (page 78)
Script Manager (page 74)
TextEdit (page 77)
Text Services Manager (page 78)
Text Utilities (page 78)

Low-Level Operating System Services

The low-level operating system services provide your application with facilities ranging from memory management to process management—essential services that are generally devoid of a user interface. Carbon supports most of the programming interfaces to these services. However, slight changes to the interfaces and implementations of these services may require you to revise portions of your code.

For example, changes to the Memory Manager may affect your application if it uses zones, system memory, or temporary memory. As another example, file system specification (`FSSpec`) records must contain a volume reference number, a parent directory, and a name in Carbon functions—the substitution of a working directory for the volume reference number, or a full or partial pathname for the name, will not work. As yet another example, the Resource Manager is supported in Carbon except in certain instances involving ROM resource functions and functions that access the resource map.

As previously stated, you should avoid directly creating and manipulating data structures used by the operating system. Instead, you should use the application programming interface for accessing these structures. For example, your application should call the File Manager function `PBGetFCBInfo` to get information about the FCB table rather than calling `LMGetFCBSPtr` and walking the table.

Apple is investigating the extent to which Carbon should support the Patch Manager, the Power Manager, and the Shutdown Manager. Please give us feedback on the importance of these interfaces in your development efforts.

For more information on the individual technologies that make up the low-level operating system services, please see the following:

Alias Manager (page 45)
Apple Event Manager (page 46)
Collection Manager (page 48)
Debugger Services (page 52)
Disk Initialization Manager (page 53)
Event Manager (page 55)
Notification Manager (page 66)
Patch Manager (page 68)
Power Manager (page 70)
PPC Toolbox (page 70)
Process Manager (page 71)
Resource Manager (page 73)

Exception Manager (page 55)	Shutdown Manager (page 75)
File Manager (page 55)	Start Manager (page 76)
Gestalt Manager (page 57)	Thread Manager (page 78)
Low Memory Accessors (page 60)	Time Manager (page 79)
Mathematical and Logical Utilities (page 61)	Memory Management Utilities (page 61)
Memory Manager (page 62)	Vertical Retrace Manager (page 79)
Multiprocessing Services (page 65)	

Networking and Communication Services

In Carbon, Open Transport supplants earlier Mac OS programming interfaces for networking and communication services. If your application uses Open Transport, the functions called by your application are likely to be supported in Carbon.

Carbon does not support the AppleTalk Manager or MacTCP. To implement networking features based on AppleTalk or TCP/IP protocols, you should use the Open Transport programming interface.

Because of limited customer adoption of the features offered by AOCE (Apple Open Collaboration Environment) and the Telephone Manager, these programming interfaces are also unsupported in Carbon.

Apple is currently reviewing the programming interface defined by the Communications Toolbox, used by some developers for their data connection, terminal emulation, and file transfer products. Please give us feedback on the importance of this interface in your development efforts.

For more information on the individual technologies that make up the networking and communication services, please see the following:

AppleTalk Manager (page 47)	Open Transport (page 66)
Communications Toolbox (page 49)	MacTCP (page 61)
Digital Signature Manager (page 53)	Telephone Manager (page 77)
Apple Open Collaboration Environment (AOCE) (page 46)	

Runtime Services

The runtime services provide programming interfaces that prepare code for execution and control how functions call each other. Carbon supports the majority of these programming interfaces.

Because Mac OS X will not run 68K code, the Mixed Mode Manager will not serve any useful purpose on that platform. In Mac OS 8, Carbon fully supports the Mixed Mode Manager. Carbon also supports universal procedure pointers (UPPs) transparently, so you do not have to change them or remove them from your code.

Carbon doesn't support the Package Manager, which relies on 68K code. Segment loading and unloading is specific to Classic 68K and CFM-68K runtime architectures, and therefore the Segment Manager is also unsupported. Apple is not planning to support the Trap Manager in Carbon, because traps are used only in 68K code. So that we can plan a suitable replacement, please give us feedback on how you are using the Trap Manager.

For more information on the individual technologies that make up the runtime services, please see the following:

Code Fragment Manager (page 48)
Component Manager (page 50)
Data Types (page 51)
JManager (page 59)
Mixed Mode Manager (page 65)
Package Manager (page 68)
Segment Manager (page 74)
System Error Handler (page 77)
Trap Manager (page 79)

Hardware Interfaces

The hardware interfaces provide low-level access to hardware, such as keyboards, PCI cards, and SCSI devices. To enhance system stability and I/O efficiency, Mac OS X will carefully control access to hardware devices. These enhancements necessitate changes to many of the hardware interfaces.

In general, if your application needs to interact directly with hardware devices, you'll likely need to modify your code. Carbon supports application-level access to ADB, SCSI, and serial devices. However, other hardware devices won't be accessible from application-level software in Mac OS X.

In Mac OS X, access to most hardware devices will be restricted to driver-level code running in the microkernel context. Apple is developing a new device driver programming interface, called the IOKit, for writing Mac OS X device drivers. Whenever possible, the functions in the IOKit will be identical to those of the programming interfaces it replaces. The IOKit will also provide an API for applications to communicate with device drivers. If your software product requires application-level programming interfaces instead of driver-level interfaces, please give us feedback about your needs.

Because the IOKit is specific to Mac OS X, your application will continue to use the existing programming interfaces when running in Mac OS 8.

For more information on the individual technologies that make up the hardware interfaces, please see the following:

ADB Manager (page 45)
ATA Manager (page 47)
Block-Level Device Drivers (page 48)
Cursor Devices (page 51)
Device Manager (page 52)
Ethernet Driver (page 54)
Name Registry (page 65)
PC Card Services (page 68)
PCI Card Services (page 69)
SCSI Manager (page 74)
Serial Driver (page 75)
Slot Manager (page 75)

MPW and Related Interfaces

The Macintosh Programmer's Workshop (MPW) provides programming interfaces that aren't part of the Mac OS system software. However, Apple plans to make these interfaces compatible with Carbon. For more information about the individual interfaces, please see the following:

ANSI C Interfaces (page 81)
C++ Interfaces (page 82)
MPW C Interfaces (page 82)
MPW Utilities (page 82)

C H A P T E R 3

Preparing for Mac OS X

Contents

This chapter describes two things you can do right away to begin preparing for Mac OS X:

- Use the tool described in the next section to analyze your applications and libraries for compatibility with Carbon.
- Follow the programming guidelines described in "Programming for Mac OS X" (page 38).

We want to make it as easy as possible for you to prepare your application for Mac OS X by adopting Carbon. We welcome your feedback on the compatibility analyzer, the compatibility reports, and the programming guidelines described in this chapter. Talk to us at the Worldwide Developers Conference (WWDC) Compatibility Lab or send your comments and suggestions to this e-mail address:

carbon@apple.com

The Carbon Compatibility Analyzer

Apple is providing a tool to analyze compiled applications and libraries for compatibility with Carbon. You can use this tool to obtain information about the compatibility of your existing code and the scope of your future conversion efforts. We encourage you to bring your applications to the WWDC Compatibility Lab for testing. After WWDC, Apple will make the compatibility analyzer available at

<http://developer.apple.com/macosx/>

The Analyzer

The compatibility analyzer examines compiled applications and libraries and generates an output file that lists the Mac OS functions your code calls. This file is used to create the final compatibility report for your application. The analyzer obtains function information from the Mac OS Universal Interfaces, currently at version 3.1. The output file may also contain information about other potential compatibility issues, such as direct access to low memory locations.

IMPORTANT

The compatibility analyzer will not expose any of your proprietary information. The compatibility analyzer reports only on calls to Mac OS functions and on certain other potential compatibility issues. You can examine the output file to verify its contents. ▲

The Report

The output file created by the compatibility analyzer is used to create a Carbon compatibility report for your application. The compatibility report is an HTML document that provides a snapshot of the latest Carbon compatibility information for your application.

Mac OS Functions

For each Mac OS function your code calls that is not fully supported in Carbon, the compatibility report specifies whether the function is

- supported but modified in some way from how it is used in previous versions of the Mac OS
- supported but not recommended—that is, you can use the function, but it may not be supported in the future
- unsupported
- under evaluation—Apple has made no decision on whether to support the function. If you need to use any of these functions, let us know why.
- not found in the Universal Interfaces 3.1

The report includes a chart that shows the percentages of Mac OS functions in each category. For many functions, the report also describes how to modify your application. For example, text accompanying an unsupported function might describe a replacement function or recommended workaround.

Direct Access to Low Memory

The compatibility report lists instances where your code makes a direct access to low memory. For information on how to access low memory correctly, see "Low Memory Accessors" (page 60). If the tested code was built with symbolic debugging information enabled, the report specifies the names of the routines that access low memory directly.

Note
When you use an Apple-supplied low-memory accessor routine, it may be inserted inline in your code. As a result, the analyzer may flag instances where you access low memory with an approved accessor. ◆

Resources Stored in the System Heap

The compatibility report lists resources that have their system heap bit set, indicating they should be stored in the system heap. For each flagged resource, the report lists the resource type and ID, as well as the resource name if one is available. Note that you will not have access to the system heap in Mac OS X as you do in previous versions of the Mac OS. For more information on the system heap, see "Memory Manager" (page 62).

Additional Reports

As work on Mac OS X and Carbon progresses, it is likely that there will be changes in the planned support for some functions. As a result, you should occasionally get new reports based on your and Apple's most recent efforts. For information on how to obtain additional reports, see

<http://developer.apple.com/macosx/>

IMPORTANT
The compatibility analyzer cannot guarantee that your application is entirely compatible with Carbon and Mac OS X, even if your report lists no specific incompatibilities. For example, applications might access low memory in a way that is not supported but that cannot be detected by the compatibility analyzer. ▲

Programming for Mac OS X

This section provides guidelines to help you prepare your application for Mac OS X. Most PowerPC-native applications that conform to Macintosh development guidelines should require relatively little modification. Many of these guidelines will be familiar to Mac OS developers.

- Use the latest Universal Interfaces (currently version 3.1), which are available at

  ftp://dev.apple.com/devworld/Development_Kits/Interfaces-Libraries3.1.sit.hqx
- Link against Apple-provided libraries for all system-supplied interfaces.
- Apple encourages you to replace the Standard File Package with Navigation Services in your current Mac OS development efforts. Navigation Services Version 1.0 is compatible with system software versions back to System 7.5.5. To assist in the switch to Navigation Services, Apple will provide source code that maps many common Standard File calls to their Navigation Services counterparts. Note, however, that this code provides an interim solution only—you will eventually need to adopt Navigation Services, because Carbon won't support the Standard File Package.
- Adopt the Open Transport programming interfaces, which will replace the AppleTalk Manager and MacTCP in Mac OS X.
- Prepare for protected memory, as described in "Memory Manager" (page 62).
- Avoid accessing low memory variables, as described in "Low Memory Accessors" (page 60).
- Do not rely on internal data structures or the layout of unpublished structures.
- Use only the functions defined by the Human Interface Toolbox managers to create and dispose of Toolbox-related data structures. For more information on these functions, see "Data Structure Access" (page 24).
- In some cases, Apple will supply source code for glue that maps an unsupported function call to a supported call—for example, this glue might map the `setstring` function to `SetString`. You can compile this source code into your application to provide an interim compatibility solution.

- Do not use 68K code—it won't be supported in Mac OS X. For example, if your application contains any libraries that are packaged as 68K code resources, you will have to recompile them for PowerPC.
- Do not write code that directly accesses hardware—use programmatic interfaces for interacting with sound, serial, and other hardware interfaces.
- Do not patch traps. If you feel you must, please tell Apple what functionality you would like us to provide to take the place of your patch.
- Use system services for timing.
- Locate special folders with the function `FindFolder`.

If your code doesn't adhere to these guidelines, you should consider updating it during your next development cycle. The effort you put in now will help greatly in making your application compatible with Mac OS X. Your effort will also ensure greater compatibility with future operating system changes.

C H A P T E R 4

Carbon Reference

Contents

Carbon Reference

This chapter describes the programming interfaces that Apple currently distributes to developers and the planned level of Carbon support for each interface. The header files associated with each interface are also listed.

IMPORTANT
Only system software interfaces that are part of Universal Interfaces 3.1 are being considered for inclusion in Carbon. ▲

Universal Interfaces

This section describes the level of Carbon support planned for existing Mac OS interfaces contained in Universal Interfaces 3.1. Interfaces are listed alphabetically.

Note
Not all header files contain functions; some contain only data structures or constants. ◆

For more information about changes and workarounds for functions associated with a given interface, you can access the current Carbon compatibility specification through the following web site:

<http://developer.apple.com/macosx/>

ADB Manager

Carbon will fully support the ADB Manager. The functions, structures, and constants of the ADB Manager appear in the following interface file:

`Deskbus.h`

Alias Manager

Carbon will fully support the Alias Manager. The functions, structures, and constants of the Alias Manager appear in the following interface file:

`Aliases.h`

Apple Open Collaboration Environment (AOCE)

Carbon will not support the Apple Open Collaboration Environment because this interface is not part of Mac OS 8. The functions, structures, and constants of AOCE appear in the following interface files:

```
OCE.h                          OCEStandardDirectory.h
OCEAuthDir.h                   OCEStandardMail.h
OCEMail.h                      OCETemplates.h
OCEMessaging.h
```

Appearance Manager

Carbon will fully support the Appearance Manager. The functions, structures, and constants of the Appearance Manager appear in the following interface file:

```
Appearance.h
```

Apple Event Manager

Carbon will fully support the Apple Event Manager. Standard Apple events previously defined by Apple—for example, the required suite of Apple events discussed in *Inside Macintosh: Interapplication Communication*—will be supported in Mac OS X and play the same roles as they do in Mac OS 8. See also "Open Scripting Architecture" (page 66) for information on scripting support.

The functions, structures, and constants of the Apple Event Manager appear in the following interface files:

```
AppleEvents.h                  AEPackObject.h
AEDataModel.h                  AERegistry.h
AEObjects.h                    AEUserTermTypes.h
```

Apple Guide Manager

Carbon will fully support the Apple Guide Manager. The functions, structures, and constants of Apple Guide appear in the following interface file:

```
AppleGuide.h
```

AppleTalk Manager

Carbon will not support the AppleTalk Manager. You should instead use Open Transport to implement networking features based on AppleTalk protocols.

The functions, structures, and constants of the AppleTalk Manager appear in the following interface files:

`ADSP.h`

`ADSPSecure.h`

`AppleTalk.h`

Apple Type Solution for Unicode Imaging

The Apple type solution for Unicode imaging (ATSUI) will be introduced soon. Carbon will fully support this new programming interface. The functions, structures, and constants of ATSUI will appear in the following interface files:

`ATSUnicode.h`

`Unicode.h`

ATA Manager

Carbon will not support the ATA Manager. In Mac OS X, code that communicates directly with ATA devices must use the IOKit API.

The functions, structures, and constants of the ATA Manager appear in the following interface file:

`ATA.h`

AV Components

Carbon will not support the AV components programming interface. This interface allowed developers to extend the Monitors and Sound control panel. Apple will define a new programming interface to provide similar functionality in Mac OS X.

The functions, structures, and constants of AV Components appear in the following interface file:

`AVComponents.h`

Block-Level Device Drivers

Carbon will not support direct access to block storage devices from application-level code. In Mac OS X, code that communicates directly with block-level devices must use the IOKit API. Please give us feedback if you see a need to call these functions from application-level code.

The functions, structures, and constants of the block-level device drivers appear in the following interface file:

`Disks.h`

Code Fragment Manager

Carbon will fully support the Code Fragment Manager. The functions, structures, and constants of the Code Fragment Manager appear in the following interface file:

`CodeFragments.h`

Note
`CodeFragments.h` replaces the older file `FragLoad.h`. ◆

Collection Manager

Carbon will fully support the Collection Manager. The functions, structures, and constants of the Collection Manager appear in the following interface file:

`Collections.h`

Color Picker Manager

Carbon will support the Color Picker Manager functions that most applications rely on, `GetColor`, `PickColor`, and `NPickColor`. In addition, Carbon will support all the functions described as supported in “Technote 1100: Color Picker 2.1.” That same Technote specifies certain functions from earlier versions of the Color Picker Manager that are no longer supported by the Mac OS; Carbon will not support these functions.

The functions, structures, and constants of the Color Picker Manager appear in the following interface files:

`ColorPicker.h`	`Picker.h`
`ColorPickerComponents.h`	

ColorSync Manager

Carbon will support the majority of the ColorSync Manager programming interface. However, ColorSync 1.0 compatibility calls such as `CWNewColorWorld`, `GetProfile`, and `SetProfile` will not be supported. We welcome feedback on the importance of these routines in your development efforts.

Some applications use the Component Manager to determine what color management modules (CMMs) are available. You will not be able to use the Component Manager for this purpose in Mac OS X, but Apple will provide a new API to query for available CMMs. We are investigating the runtime model for CMMs; it is possible that CMM developers will have to rewrite their code to replace the Component Manager interface layer.

The functions, structures, and constants of the ColorSync Manager appear in the following interface files:

`CMAcceleration.h`	`CMMComponent.h`
`CMApplication.h`	`CMPRComponent.h`
`CMConversions.h`	`CMICCProfile.h`
`CMCalibrator.h`	`CMScriptingPlugin.h`

Communications Toolbox

Apple is investigating the extent to which Carbon should support the Communications Toolbox. Please give us feedback on the importance of this interface in your development efforts.

The functions, structures, and constants of the Communications Toolbox appear in the following interface files:

`CommResources.h`	`FileTransfers.h`
`Connections.h`	`FileTransferTools.h`
`ConnectionTools.h`	`Terminals.h`
`CRMSerialDevices.h`	`TerminalTools.h`
`CTBUtilities.h`	

Component Manager

Carbon will fully support the Component Manager. The functions, structures, and constants of the Component Manager appear in the following interface file:

`Components.h`

Control Manager

Carbon will fully support the Control Manager. Be aware, however, that if you use custom control definition procedures (also known as **CDEFs**), you must compile them as PowerPC-native code.

To ensure that your application can most easily take advantage of future enhancements to the Mac OS, Apple suggests that you consider the following points.

- Your application should use the functions defined by the Control Manager whenever it creates and disposes of Control Manager data structures. For example, instead of directly creating and disposing of control records, applications should call the Control Manager functions `GetNewControl` and `DisposeControl`. In Mac OS X, applications might not be allowed to create and dispose of Control Manager data structures except by calling Control Manager functions.
- You should revise your application so that it accesses the data structures defined by the Control Manager only through accessor functions. The use of these accessor functions, to be provided soon, will give your application greater threading flexibility in Mac OS X.
- You are encouraged to adopt the standard Mac OS 8 control definition procedures in your application. Applications that use the standard control definition procedures inherit the Mac OS 8 human interface appearance. Applications that use custom control definition procedures work correctly, but because custom definition procedures invoke their own drawing routines, Mac OS 8 can't draw these applications with the current appearance.

The functions, structures, and constants of the Control Manager appear in the following interface file:

`Controls.h`

Control Strip Services

Carbon will fully support Control Strip services. However, Control Strip modules written as 68K `'sdev'` resources must be rebuilt as Control Strip 2.0 PowerPC-native modules.

The functions, structures, and constants of the Control Strip Manager appear in the following interface file:

`ControlStrip.h`

Cursor Devices

Apple is investigating the extent to which Carbon should support Cursor Devices. We anticipate providing support for the majority of the functions in this interface.

The functions, structures, and constants of Cursor Devices appear in the following interface file:

`CursorDevices.h`

Data Access Manager

Carbon will not support the Data Access Manager. This interface is not widely used. Some developers use Macintosh ODBC instead to provide a standard database API, but there is no guarantee that ODBC will be supported in Mac OS X.

The functions, structures, and constants of the Data Access Manager appear in the following interface file:

`DatabaseAccess.h`

Data Types

Current Mac OS data types will remain unchanged in Carbon. The Mac OS data types are defined in the following interface file:

`MacTypes.h`

Note
`MacTypes.h` replaces the older file `Types.h`. ◆

Date, Time, and Measurement Utilities

Carbon will fully support the Date, Time, and Measurement Utilities. However, some obsolete functions that are prefixed with `IU` (such as `IUDateString` and `IUTimeString`) may not be supported in the future.

The functions, structures, and constants of the Date, Time, and Measurement Utilities appear in the following interface file:

`DateTimeUtils.h`

Debugger Services

Carbon will fully support the debugger services. The functions, structures, and constants of the debugger services appear in the following interface file:

`MacTypes.h`

Note
`MacTypes.h` replaces the older file `Types.h`. ◆

Device Manager

Carbon will not support the Device Manager. In Mac OS X, code that communicates directly with hardware devices must use the IOKit API. Other types of code that have relied on the Device Manager interface in the past (such as desk accessories) should be converted into applications.

The functions, structures, and constants of the Device Manager appear in the following interface file:

`Devices.h`

Dialog Manager

Carbon will fully support the Dialog Manager. However, Apple encourages you to revise your application so that it accesses the data structures defined by the Dialog Manager only through accessor functions. The use of these accessor functions, to be provided soon, will give your application greater threading flexibility in Mac OS X. Furthermore, you are encouraged to use only the functions defined by the Dialog Manager to create and dispose of data structures defined by the Dialog Manager. In Mac OS X, applications might not be allowed to create and dispose of Dialog Manager data structures except by calling Dialog Manager functions.

The functions, structures, and constants of the Dialog Manager appear in the following interface file:

```
Dialogs.h
```

Dictionary Manager

Carbon will not support the Dictionary Manager, which handles data used in text conversion for 2-byte script systems. However, Apple will provide a new Carbon-compliant Dictionary Manager in the near future. The functions, structures, and constants of the Dictionary Manager appear in the following interface file:

```
Dictionary.h
```

Digital Signature Manager

As with other AOCE technologies, Carbon will not support the Digital Signature Manager. The functions, structures, and constants of the Digital Signature Manager appear in the following interface file:

```
DigitalSignature.h
```

Disk Initialization Manager

Carbon will fully support the Disk Initialization Manager. The functions, structures, and constants of the Disk Initialization Manager appear in the following interface file:

```
DiskInit.h
```

Display Manager

Carbon will support the Display Manager, with the exception of the function `DMGetDisplayMgrA5World` (which isn't relevant in a PowerPC execution environment). The functions, structures, and constants of the Display Manager appear in the following interface file:

```
Displays.h
```

Drag Manager

Carbon will fully support the Drag Manager. The functions, structures, and constants of the Drag Manager appear in the following interface file:

```
Drag.h
```

Edition Manager

Due to limited customer adoption of the publish-and-subscribe feature, the Edition Manager will for the most part not be supported in Carbon. However, Apple will provide a mechanism for your application to open and read files that contain editions.

The functions, structures, and constants of the Edition Manager appear in the following interface file:

```
Editions.h
```

Ethernet Driver

Carbon will not support the Ethernet driver. In Mac OS X, code that communicates directly with Ethernet hardware must use the IOKit API.

The functions, structures, and constants of the Ethernet driver interface appear in the following interface file:

```
ENET.h
```

Event Manager

Carbon will fully support the Event Manager. The functions, structures, and constants of the Event Manager appear in the following interface files:

`EPPC.h`	`Events.h`

Note
`Events.h` replaces the older file `OSEvents.h`. ◆

Exception Manager

Carbon will fully support the Exception Manager. The functions, structures, and constants of the Exception Manager appear in the following interface file:

`MachineExceptions.h`

File Manager

While Carbon will support most of the File Manager interface, there are a number of significant changes.

You should not access File Manager structures directly if accessor functions for the structures exist. For example, you should call `PBGetFCBInfo` rather than calling `LMGetFCBSPtr` and walking the FCB table.

Similarly, you should create file system specification (`FSSpec`) records using the functions `PBMakeFSSpec` or `FSMakeFSSpec` instead of filling in your own structures. File system specification records must contain a volume reference number, a parent directory, and a name. Substituting a working directory for the volume reference number or a full or partial pathname for the name is not supported.

File Manager functions that support MFS (the Macintosh file system) will not be supported in Carbon. These include

- functions, such as `OpenWD` and `GetWDInfo`, that manipulate working directories
- functions that identify a file or folder using a volume reference number and pathname but not a parent directory ID. You should use the HFS version of these calls (which use a parent directory ID) instead.

Functions that cannot be called by PowerPC applications (such as `PBGetAltAccessSync` and `PBGetAltAccessAsync`) will not be supported.

In general, you should use only documented File Manager interfaces in your application. For example, you should use a documented API call to get low memory information rather than access undocumented low memory global variables or vectors directly.

The default volume format for Carbon will be the Mac OS Extended format; in most cases you will not need to modify your application to gain this volume support.

Carbon may not support the File System Manager (FSM) as defined in the header file `FSM.h`. FSM plug-ins today often must make assumptions about File Manager data structures that may not be valid in Carbon. In addition, some plug-ins use 68K calling conventions when calling internal File Manager functions. Since the File System Manager was designed for a single-threaded file system, it cannot use the full threading capabilities of Mac OS X.

The functions, structures, and constants of the File Manager appear in the following interface files:

```
Files.h                           FSM.h
FileTypesAndCreators.h            HFSVolumes.h
```

Finder Interface

Carbon will fully support the Finder Interface. The structures and constants of the Finder Interface appear in the following interface files:

```
Finder.h                          FinderRegistry.h
```

Folder Manager

Carbon will fully support the Folder Manager. The functions, structures, and constants of the Folder Manager appear in the following interface file:

```
Folders.h
```

Font Manager

Carbon will fully support the Font Manager. The functions, structures, and constants of the Font Manager appear in the following interface files:

`Fonts.h`	`SFNTTypes.h`
`SFNTLayoutTypes.h`	

Game Sprockets

Apple is investigating the extent to which Carbon should support Game Sprockets. Please give us feedback on the importance of this interface in your development efforts.

The functions, structures, and constants of Game Sprockets appear in the following interface files:

`DrawSprocket.h`	`InputSprocket.h`
`GoggleSprocket.h`	`NetSprocket.h`
`GoggleSprocketDevices.h`	`SoundSprocket.h`

Gestalt Manager

Carbon will fully support the Gestalt Manager. However, you should not use Gestalt functions to pass pointers to data among applications because each application will reside in its own protected memory space.

You should avoid using the `NewGestalt` function to add a selector code, which requires moving your selector function into the system heap. You should also avoid using the `ReplaceGestalt` function to replace an existing selector function, which also requires your replacement function to reside in the system heap. Note that you will not have access to the system heap in Mac OS X as you do in previous versions of the Mac OS

The functions, structures, and constants of the Gestalt Manager appear in the following interface file:

`Gestalt.h`

Note
`Gestalt.h` replaces the older file `GestaltEqu.h`. ◆

Help Manager

Carbon will support the majority of the programming interface defined by the Help Manager. However, there are portions of this programming interface that won't be supported.

- So that help balloons keep the systemwide appearance, applications won't be able to set the font or font size for balloon text.
- The Help Manager won't allow applications to display pictures in help balloons.
- The Help Manager won't support help balloons for custom menu definition functions.

The functions, structures, and constants of the Help Manager appear in the following interface file:

```
Balloons.h
```

HyperCard XCommand Support

Carbon will fully support HyperCard XCommands. Note that the functions in this programming interface are not part of Mac OS system software—they are implemented in the HyperCard application.

The functions, structures, and constants for HyperCard XCommands appear in the following interface file:

```
HyperXCmd.h
```

Icon Utilities

Carbon will fully support Icon Utilities. The functions, structures, and constants of Icon Utilities appear in the following interface file:

```
Icons.h
```

Image Compression Manager

Carbon will fully support the Image Compression Manager. The functions, structures, and constants of the Image Compression Manager appear in the following interface files:

```
ImageCodec.h                          ImageCompression.h
```

International Resources

Carbon will fully support the International Resources. However, if your application uses 68K code in these resources, you must compile them as PowerPC-native code. The structures and constants of International Resources appear in the following interface files:

```
IntlResources.h                       WorldScript.h
```

JManager

While Carbon will support most of the JManager interface, functions that use the Java Runtime Interface (JRI) will not be supported. You should use corresponding functions that use the Java Native Interface (JNI) instead.

The functions, structures, and constants of JManager appear in the following interface file:

```
JManager.h
```

List Manager

Carbon will fully support the List Manager. Applications that use custom list definition procedures (also known as **LDEFs**) work correctly; however, you must compile them as PowerPC-native code.

Apple encourages you to revise your application so that it accesses the data structures defined by the List Manager only through accessor functions. The use of these accessor functions, to be provided soon, will give your application greater threading flexibility in Mac OS X. Furthermore, you are encouraged to use only the functions defined by the List Manager to create and dispose of data structures defined by the List Manager. In Mac OS X, applications might not be allowed to create and dispose of List Manager data structures except by calling List Manager functions.

The functions, structures, and constants of the List Manager appear in the following interface file:

`Lists.h`

Location Manager

Carbon will fully support the Location Manager. The functions, structures, and constants of the Location Manager appear in the following interface file:

`LocationManager.h`

Low Memory Accessors

Carbon will support most of the accessor functions for low-memory variables. However, you should always avoid using low-memory accessors if there are direct Mac OS Toolbox calls to obtain the same information. For example:

- Use the function `TickCount` instead of the low-memory accessor function `LMGetTicks`.
- Use the function `FrontWindow` instead of the low-memory accessor function `LMGetWindowList`, when possible.
- Use the function `PBGetFCBInfo` instead of walking the FCB table with the low-memory accessor function `LMGetFCBSPtr`.

In general, don't think of the values returned by low-memory accessor routines as residing in low memory—think of them as information, possibly associated with a specific Toolbox manager, that is returned by the Mac OS. In the future, Apple may supply new functions, distributed among the Mac OS Toolbox managers, for retrieving this information.

Some low-memory accessor functions are likely to become obsolete in Mac OS X. The following Resource Manager–related functions have been identified as among those that will not be supported:

- `LMGetTopMapHndl` and `LMSetTopMapHndl`
- `LMGetSysMapHndl` and `LMSetSysMapHndl`
- `LMGetCurMapHndl` and `LMSetCurMapHndl`

See "Resource Manager" (page 73) for more information on ROM-related functions.

The functions, structures, and constants of the Low Memory accessors appear in the following interface file:

`LowMem.h`

MacTCP

Carbon will not support MacTCP. You should instead use Open Transport to implement networking features based on the TCP/IP protocol suite.

The functions, structures, and constants of MacTCP appear in the following interface file:

`MacTCP.h`

Mathematical and Logical Utilities

Carbon will fully support the Mathematical and Logical Utilities. The functions, structures, and constants of the Mathematical and Logical Utilities appear in the following interface files:

```
fenv.h                        Math64.h
FixMath.h                     ToolUtils.h
fp.h
```

Memory Management Utilities

While Carbon will support most of the Memory Management Utilities, there are changes to functions that assume a 68K runtime environment.

- Functions that flush caches on 68K processors (such as `FlushInstructionCache`, `FlushDataCache`, and `FlushCodeCacheRange`) will no longer be supported.
- Functions such as `SetA5` or `SetCurrentA5` will do nothing when running in Mac OS X. However, these functions should work normally when running in Mac OS 8.
- The functions `GetMMUMode` and `SwapMMUMode` will not be supported because all PowerPC applications use 32-bit addressing, even if they are not Carbon-compliant.

- The `SysEnvirons` function will no longer be supported since the Gestalt Manager can provide the same information. You should call the functions `FindFolder` and `Gestalt` instead.

The functions, structures, and constants of the Memory Management Utilities appear in the following interface file:

`OSUtils.h`

Memory Manager

Carbon will support the majority of the Memory Manager programming interface. Changes will primarily affect applications that use zones, system memory, and temporary memory. For example, temporary memory allocations in Mac OS X will be allocated in the application's address space. As a result, calling `TempNewHandle` will effectively be the same as calling `NewHandle`.

Carbon will not support current functions for accessing the system heap, but Apple will provide routines to allocate shared and persistent memory. In addition, the virtual memory system in Mac OS X will introduce a number of changes in the addressing model.

The following Memory Manager functions are likely to change in Carbon: `GZSaveHnd`, `MoreMasters`, and `TempNewHandle`.

The following Memory Manager functions are still under evaluation: `GetZone`, `HandleZone`, `InitZone`, `PtrZone`, `SetGrowZone`, and `SetZone`. Please give us feedback on the importance of using subzones in your applications.

The following Memory Manager functions will not be supported in Carbon: `MaxBlockSys`, `MaxMemSys`, `NewEmptyHandleSys`, `NewHandleSys`, `NewHandleSysClear`, `NewPtrSys`, `NewPtrSysClear`, `PurgeMemSys`, `PurgeSpaceSysContiguous`, `PurgeSpaceSysTotal`, `ReallocateHandleSys`, `RecoverHandleSys`, `RecoverMemSys`, and `SystemZone`.

The following Virtual Memory functions are likely to change in Carbon: `FlushMemory`, `MakeMemoryNonResident`, `MakeMemoryResident`, and `ReleaseMemoryData`.

By adhering to the following guidelines, you can increase your application's compliance with Mac OS X memory management:

- Try to use memory only within your own application heap. Review the places where you allocate memory in the system heap.

- Do not pass pointers to data among applications through Apple events, Gestalt routines, or other means. For example, don't share Toolbox data structures between applications, because in Mac OS X, each application will run in its own protected address space.
- Do not use inline, hard-coded addresses.
- Do not use the following Memory Manager functions: `EnterSupervisorMode`, `StripAddress`, `Translate24To32`.
- Do not modify a field in a zone header to specify how many master pointer blocks are allocated by each call to the functions `MoreMasters`. The Memory Manager will supply a new function to safely specify the number of master pointer blocks to allocate. Note, however, that master pointer blocks do not need to be preallocated or optimized in the Mac OS X memory model, so the new function will provide the most benefit for applications running on previous versions of the Mac OS.
- Don't make assumptions about the layout of memory, such as the relative position of data stored in heaps, stacks, and other memory areas.
- Don't use the `DisposePtr` or `DisposeHandle` functions on pointers or handles allocated by Toolbox managers. For example, don't call the function `DisposeHandle` on a control allocated with the function `NewControl`—use `DisposeControl` instead.
- Because Mac OS X applications will run in a large, protected memory space, memory sizing routines such as `MaxMem` and `FreeMem` will work differently than they do now.

See "Memory Management Utilities" (page 61) for information on related memory management interfaces.

The functions, structures, and constants of the Memory Manager appear in the following interface file:

```
MacMemory.h
```

Note

`MacMemory.h` replaces the older file `Memory.h`. ◆

Menu Manager

Carbon will fully support the Menu Manager. To ensure that your application can most easily take advantage of future enhancements to the Mac OS, Apple suggests that you consider the following points:

- You are strongly encouraged to adopt the standard Mac OS 8 menu definition procedures (also known as **MDEFs**) in your application. Your menus will then inherit the systemwide appearance and, furthermore, take advantage of other Menu Manager enhancements planned for the future. Note, however, that if you must use custom menu definition procedures, you must compile them as PowerPC-native code.
- Your application should use the functions defined by the Menu Manager whenever it creates and disposes of Menu Manager data structures. Some applications, for example, create menus by using the Resource Manager function `GetResource` (instead of the Menu Manager function `GetMenu`) and dispose of them by calling `ReleaseResource` instead of `DisposeMenu`. In Mac OS X, applications might not be allowed to create and dispose of Menu Manager data structures except by calling Menu Manager functions.
- You should revise your application so that it accesses the data structures defined by the Menu Manager only through accessor functions to be provided soon. The use of these accessor functions will give your application greater threading flexibility in Mac OS X.
- There are several Menu Manager functions that deal with manipulating menu color information tables. Apple recommends that you stop using them. The Appearance Manager generally disregards these color tables and instead uses the colors defined for the current appearance.

The functions, structures, and constants of the Menu Manager appear in the following interface file:

```
Menus.h
```

MIDI Manager

The MIDI Manager has been discontinued and therefore will not be supported in Carbon. The functions, structures, and constants of the MIDI Manager appear in the following interface file:

```
MIDI.h
```

Mixed Mode Manager

Mac OS X will not run 68K code, so the Mixed Mode Manager will no longer serve any useful purpose. However, Carbon will support universal procedure pointers (UPPs) transparently, so you do not have to change them or remove them from your code. You may want to keep Mixed Mode Manager calls in your application to maintain compatibility with the current version of the Mac OS. Mixed Mode Manager calls from Carbon applications running on Mac OS 8 will function normally.

The functions, structures, and constants of the Mixed Mode Manager appear in the following interface file:

`MixedMode.h`

Movie Toolbox

Carbon will fully support the Movie Toolbox. The functions, structures, and constants of the Movie Toolbox appear in the following interface files:

`Endian.h`	`MoviesFormat.h`
`Movies.h`	`QTML.h`

Multiprocessing Services

Carbon will fully support Multiprocessing Services. The functions, structures, and constants of Multiprocessing Services appear in the following interface file:

`Multiprocessing.h`

Note
`Multiprocessing.h` replaces the older file `MP.h`. ◆

Name Registry

Carbon will not support the Name Registry. In Mac OS X, the IOKit will provide similar functionality for device drivers.

The functions, structures, and constants of the Name Registry appear in the following interface file:

`NameRegistry.h`

Navigation Services

Carbon will fully support Navigation Services. Apple strongly encourages you to replace the Standard File Package with Navigation Services in your current Mac OS development efforts; Carbon won't support the Standard File Package.

To assist in the transition, Navigation Services 1.0 is compatible back to System 7.5.5. Apple will also provide source code that maps many Standard File Package calls to their Navigation Services counterparts. Note, however, that this code provides an interim solution only—you will eventually need to adopt Navigation Services.

The functions, structures, and constants of Navigation Services appear in the following interface file:

```
Navigation.h
```

Notification Manager

Carbon will fully support the Notification Manager. The functions, structures, and constants of the Notification Manager appear in the following interface file:

```
Notification.h
```

Open Scripting Architecture

Carbon will fully support the Open Scripting Architecture (OSA), including AppleScript-specific interfaces. See "Apple Event Manager" (page 46) for related information.

The functions, structures, and constants of OSA appear in the following interface files:

```
AppleScript.h                   OSAComp.h
ASRegistry.h                    OSAGeneric.h
OSA.h
```

Open Transport

Apple has identified the Open Transport functions, structures, and constants that are most commonly used by a large number of applications and by Mac OS system software. Carbon will support this key subset of the current interface. As a result, the Open Transport functions your application uses are likely to be available in Mac OS X. If you wish to use an Open Transport function that is not supported, please let us know.

Carbon will support the most important Open Transport endpoint routines—those that support the AppleTalk and TCP protocols. These routines are defined in the header files `OpenTransport.h`, `OpenTptAppleTalk.h`, and `OpenTptInternet.h`. We are investigating whether to support other endpoint routines.

Many programs use the AppleTalk Manager to implement a simple network serial number registration scheme. Apple will provide sample code for such simple NBP operations on a future developer CD. For more sophisticated uses of the AppleTalk Manager, you should visit the Open Transport web site:

<http://devworld.apple.com/dev/opentransport/>

which has extensive references to OT documentation and samples. Specifically, you should read *Inside Macintosh: Networking with Open Transport* (revised for Open Transport version 1.3), now available in HTML and PDF format from the above site.

You may have to revise your code if it uses Open Transport in one of the following ways:

■ Your application uses a function that directly gains access to a network port. In Mac OS X, code that communicates directly with network interfaces must use the IOKit API.

■ Your application uses the connection-oriented transaction-based endpoint feature of Open Transport. This feature will not be supported in Carbon. Removal of this capability should affect only users of AppleTalk protocols such as ASP. Note that the Mac OS does not currently have a native ASP implementation.

■ Your application uses Open Transport's XTI interfaces or UNIX® stream interfaces. Due to limited customer adoption, Carbon will not support these interfaces.

The functions, structures, and constants of Open Transport appear in the following interface files:

```
cred.h                        OpenTptModule.h
dlpi.h                        OpenTptPCISupport.h
modnames.h                    OpenTptSerial.h
OpenTptAppleTalk.h            OpenTptXTI.h
```

`OpenTptClient.h`	`OpenTransport.h`
`OpenTptCommon.h`	`strlog.h`
`OpenTptDevLinks.h`	`stropts.c`
`OpenTptInternet.h`	`strstat.h`
`OpenTptLinks.h`	`tihdr.h`

Package Manager

Mac OS X will not run 68K code, so Carbon will not support the Package Manager, which assumes 68K code packages stored in resources.

The functions, structures, and constants of the Package Manager appear in the following interface file:

`Packages.h`

Palette Manager

Carbon will fully support the Palette Manager. The functions, structures, and constants of the Palette Manager appear in the following interface file:

`Palettes.h`

Patch Manager

It is unlikely that the Patch Manager will be supported in Carbon. It is important that you tell us why you use this manager so we can plan a suitable replacement. For example, if you currently patch `ExitToShell`, you can instead use a new callback mechanism that provides the same function.

The functions, structures, and constants of the Patch Manager appear in the following interface file:

`Patches.h`

PC Card Services

Carbon will not support PC card services. In Mac OS X, code that communicates directly with PC cards must use the IOKit API. Please give us feedback if you see a need to call these functions from application-level code.

The functions, structures, and constants of PC card services appear in the following interface files:

`CardServices.h`	`PCCardTuples.h`
`PCCard.h`	`SocketServices.h`
`PCCardAdapterPlugin.h`	`ZoomedVideo.h`
`PCCardEnablerPlugin.h`	

PCI Card Services

Carbon will not support PCI card services. In Mac OS X, all code that communicates directly with PCI cards must use the IOKit API. Please give us feedback if you see a need to call these functions from application-level code.

The functions, structures, and constants of PCI card services appear in the following interface files:

`DriverFamilyMatching.h`	`DriverSynchronization.h`
`DriverGestalt.h`	`PCI.h`
`DriverServices.h`	`Video.h`
`DriverSupport.h`	`VideoServices.h`

Note
The functionality of the older files `Interrupts.h` and `Kernel.h` are now contained in `DriverServices.h`. ◆

Picture Utilities

Carbon will fully support the Picture Utilities. The functions, structures, and constants of the Picture Utilities appear in the following interface file:

`PictUtils.h`

Note
`PictUtils.h` replaces the older file `PictUtil.h`. ◆

Power Manager

Apple is investigating the extent to which Carbon should support the Power Manager. Please give us feedback on the importance of this interface in your development efforts.

The functions, structures, and constants of the Power Manager appear in the following interface file:

```
Power.h
```

PPC Toolbox

Carbon will fully support the PPC Toolbox. The functions, structures, and constants of the PPC Toolbox appear in the following interface file:

```
PPCToolbox.h
```

Printing Manager

Carbon will support the high-level programming interface defined by the Printing Manager. Carbon will, however, make the following changes to your use of this interface:

- The print (`TPrint`) record will no longer be directly accessible to your application. In place of a print record, the Printing Manager will provide your application with a reference to a privately defined data structure. Access to other Printing Manager data structures, such as the printing graphics port (`TPrPort`), will remain unchanged in Carbon.
- The private structure replacing the print record won't necessarily be 120 bytes long—your application must not assume it to be of any specific size.
- Apple will introduce new panel-based dialog boxes for printing. Your application should limit calls to the functions `PrStlInit`, `PrJobInit`, and `PrDlgMain` for adding items to the Page Setup and Print dialog boxes.
- Apple will make print dialog boxes movable and modal. This will require you to provide an event handler to respond to update events when your application displays these dialog boxes.
- The 68K code resource mechanism for accessing drivers will be replaced by a shared library mechanism, requiring the revision of print drivers.

- All print drivers will have to support their own spooling.

The functions, structures, and constants of the Printing Manager appear in the following interface file:

`Printing.h`

Process Manager

Carbon will support most of the Process Manager interface. The exception is the `LaunchDeskAccessory` function, which may not be supported (or it may simply do nothing).

Note that some Process Manager functions access a `ProcessInfoRec` data structure, which contains fields that are no longer applicable in a preemptively scheduled environment (for example, the `processLocation`, `processFreeMem`, and `processActiveTime` fields). Your application should avoid accessing such fields. Changes to the memory model may also affect certain fields.

The functions, structures, and constants of the Process Manager appear in the following interface files:

`Processes.h`

QuickDraw

Carbon will fully support QuickDraw.

In the past, parts of QuickDraw have been documented in *Inside Macintosh* as the Color Manager and Cursor Utilities. Note, however, that *Inside Macintosh: Imaging with QuickDraw* describes animated cursor functions, such as `SpinCursor`, that are available in the MPW programming environment and possibly in other environments, but not in system software. These animated cursor functions, which are defined in the MPW interface file `CursorCtl.h`, will continue to be supported in Carbon by MPW.

The functions, structures, and constants of QuickDraw appear in the following interface files:

`QDOffscreen.h`	`QuickDraw.h`

QuickDraw 3D

Carbon will fully support QuickDraw 3D. The functions, structures, and constants of QuickDraw 3D appear in the following interface files:

```
QD3D.h
QD3DAcceleration.h
QD3DCamera.h
QD3DController.h
QD3DDrawContext.h
QD3DErrors.h
QD3DExtension.h
QD3DGeometry.h
QD3DGroup.h
QD3DIO.h
QD3DLight.h
QD3DMath.h
QD3DPick.h
QD3DRenderer.h
QD3DSet.h
QD3DShader.h
QD3DStorage.h
QD3DString.h
QD3DStyle.h
QD3DTransform.h
QD3DView.h
QD3DViewer.h
QD3DWinViewer.h
```

QuickDraw GX

Due to limited developer adoption, Carbon will not support QuickDraw GX. You should use the other Mac OS imaging and typography managers such as QuickDraw and the Font Manager. The functions, structures, and constants of QuickDraw GX appear in the following interface files:

```
GXEnvironment.h
GXErrors.h
GXFonts.h
GXGraphics.h
GXLayout.h
GXMath.h
GXMessages.h
GXPrinterDrivers.h
GXPrinting.h
GXTypes.h
ScalerStreamTypes.h
ScalerTypes.h
```

QuickDraw Text

Carbon will fully support QuickDraw Text. The functions, structures, and constants of QuickDraw Text appear in the following interface file:

```
QuickDrawText.h
```

QuickTime Components

Carbon will fully support QuickTime components. The functions, structures, and constants of QuickTime Components appear in the following interface files:

```
MediaHandlers.h                  QuickTimeVR.h
QuickTimeComponents.h            QuickTimeVRFormat.h
QuickTimeMusic.h
```

Resource Manager

Carbon will fully support the Resource Manager, with the following exceptions, which primarily involve ROM resource functions and functions that access the resource map:

- There will be no ROM in Mac OS X, so the ROM-related functions `RGetResource` and `TempInsertROMMap` will not be supported.
- There will be no way to directly walk the resource chain. Instead, Apple will supply a small number of functions for proper management of the resource chain.
- The function `RsrcMapEntry`, which provides an interface to the map handle format, will not be supported.
- The low memory global variables `JCheckLoad`, `TopMapHndl`, `CurMap`, and `SysMapHndl` will not be supported.

Note also that you should not attempt to store resources in the system heap, since you will not have the same access to the system heap as you do in previous versions of the Mac OS. For more information on the system heap, see “Memory Manager” (page 62).

The functions, structures, and constants of the Resource Manager appear in the following interface file:

```
Resources.h
```

Scrap Manager

Carbon will fully support the Scrap Manager. The functions, structures, and constants of the Scrap Manager appear in the following interface file:

```
Scrap.h
```

Script Manager

Carbon will support nearly the entire Script Manager programming interface. Two exceptions are the `GetScriptQDPatchAddress` and the `SetScriptQDPatchAddress` functions, which are rarely used by applications.

The functions, structures, and constants of the Script Manager appear in the following interface file:

```
Script.h
```

Note
`Script.h` contains the functionality of the older file `Language.h`. ◆

SCSI Manager

Carbon will support the `SCSIAction` function as documented in the chapter "SCSI Manager 4.3" of *Inside Macintosh: Devices*. Applications can use the `SCSIAction` function to send commands to SCSI devices. No other SCSI Manager functions will be accessible to application-level code in Mac OS X. The IOKit API will provide functions for supporting SCSI device drivers.

The functions, structures, and constants of the SCSI Manager appear in the following interface file:

```
SCSI.h
```

Segment Manager

Because segment loading and unloading is specific to the Classic 68K and the later CFM-68K runtime architectures, Carbon will not support the Segment Manager.

The functions, structures, and constants of the Segment Manager appear in the following interface file:

```
SegLoad.h
```

Serial Driver

Carbon will fully support the Serial Driver. The functions, structures, and constants of the Serial Driver appear in the following interface files:

`Serial.h`

Shutdown Manager

Apple is investigating the extent to which Carbon should support the Shutdown Manager. It is anticipated that most of the functions in this interface will be supported. The functions, structures, and constants of the Shutdown Manager appear in the following interface file:

`ShutDown.h`

Slot Manager

Apple is investigating the extent to which Carbon should support the Slot Manager. Please give us feedback on the importance of this interface in your development efforts.

The functions, structures, and constants of the Slot Manager appear in the following interface file:

`ROMDefs.h`	`Slots.h`

Sound Manager

Carbon will support most Sound Manager functions. The functions `SetSoundVol`, `GetSoundVol`, `SndAddModifier`, and `SndControl` were made obsolete by Sound Manager 3.0 and are replaced by other Sound Manager functions. For functions that are no longer recommended, QuickTime often provides a simpler and more flexible alternative.

The functions, structures, and constants of the Sound Manager appear in the following interface files:

`AIFF.h`	`Sound.h`

Note
`Sound.h` replaces the older files `SoundComponents.h` and `SoundInput.h`. ◆

Speech Manager

Apple is investigating the extent to which Carbon should support the Speech Manager. Please give us feedback on the importance of this interface in your development efforts.

The functions, structures, and constants of the Speech Manager appear in the following interface files:

```
SpeechRecognition.h                  SpeechSynthesis.h
```

Note
`SpeechSynthesis.h` replaces the older file `Speech.h`. ◆

Standard File Package

Carbon will not support the Standard File Package, so you should use Navigation Services instead. Navigation Services 1.0 is compatible back to System 7.5.5.

To assist your transition efforts, Apple will provide source code that maps most Standard File Package calls to their Navigation Services counterparts. Note, however, that this code provides an interim solution only—you will eventually need to adopt Navigation Services.

The functions, structures, and constants of the Standard File Package appear in the following interface file:

```
StandardFile.h
```

Start Manager

Carbon will support most of the Start Manager interface. However, functions that install or remove callback handlers (such as `InstallExtensionNotificationProc`) for files of type `'INIT'` are not supported, since Mac OS X will not support traditional extensions.

The functions, structures, and constants of the Start Manager appear in the following interface file:

`Start.h`

System Error Handler

The System Error Handler function `SysError`, which terminates a process and displays the message "The application...has unexpectedly quit," will probably not be supported in the future. If your application needs to use this service, please let us know.

The system error constants will be supported in Carbon. Mac OS functions with a return type of `OSErr` return values based on these constants.

The functions, structures, and constants defined by the System Error Handler appear in the following interface file:

`Errors.h`

Telephone Manager

Carbon will not support the Telephone Manager. The features provided by this manager are not in wide use.

The functions, structures, and constants of the Telephone Manager appear in the following interface files:

`Telephones.h`	`TelOther601.h`
`TelephoneTools.h`	

TextEdit

Carbon will fully support TextEdit. The functions, structures, and constants of TextEdit appear in the following interface files:

`TextEdit.h`	`TSMTE.h`

Text Encoding Conversion Manager

Carbon will fully support the Text Encoding Converter Manager. The functions, structures, and constants of the Text Encoding Conversion Manager appear in the following interface files:

`TextCommon.h`

`TextEncodingConverter.h`

`UnicodeConverter.h`

Text Services Manager

Carbon will fully support the Text Services Manager. However, you should replace any calls to `NewServiceWindow` with calls to `NewCServiceWindow`, as the former may not be supported in the future.

The functions, structures, and constants of the Text Services Manager appear in the following interface file:

`TextServices.h`

Text Utilities

Carbon will fully support Text Utilities. However, some obsolete functions that are prefixed with `IU` (such as `IUStringOrder` and `IUTextOrder`) may not be supported in the future.

The functions, structures, and constants of Text Utilities appear in the following interface files:

`NumberFormatting.h`

`StringCompare.h`

`TextUtils.h`

Note

`TextUtils.h` contains the functionality of the older file `Strings.h`. ◆

Thread Manager

Carbon will fully support the Thread Manager. Note, however, that the Thread Manager currently provides only cooperative threads. If you want to use preemptive threads, you should use the Multiprocessing Services interface instead.

The functions, structures, and constants of the Thread Manager appear in the following interface file:

```
Threads.h
```

Time Manager

Carbon will fully support the Time Manager. The functions, structures, and constants of the Time Manager appear in the following interface file:

```
Timer.h
```

Translation Manager

Carbon will fully support the Translation Manager. The functions, structures, and constants of the Translation Manager appear in the following interface files:

```
Translations.h                     TranslationExtensions.h
```

Trap Manager

Because traps are used only in 68K code, Carbon will probably not support the Trap Manager. Please give us feedback on how you are using the Trap Manager so we can plan a suitable replacement.

The functions, structures, and constants of the Trap Manager appear in the following interface file:

```
Traps.h
```

Vertical Retrace Manager

Apple is investigating the extent to which Carbon should support the Vertical Retrace Manager. We anticipate providing support for the majority of the functions in this interface. If not, we will provide sample code that demonstrates how to get equivalent functionality using the Time Manager.

The functions, structures, and constants of the Vertical Retrace Manager appear in the following interface file:

```
Retrace.h
```

Virtual Memory

The programming interface for virtual memory is defined in the header `MacMemory.h`. For information on the virtual memory interface, see "Memory Manager" (page 62).

Window Manager

Carbon will fully support the Window Manager. Be aware, however, that if you use custom window definition procedures (also known as **WDEFs**), you must compile them as PowerPC-native code.

To ensure that your application can most easily take advantage of future enhancements to the Mac OS, Apple suggests that you also consider the following points.

- Your application should use the functions defined by the Window Manager whenever it creates and disposes of Window Manager data structures. For example, instead of directly creating and disposing of window records, applications should call the Window Manager functions `GetNewCWindow` and `DisposeWindow`. In Mac OS X, applications might not be allowed to create and dispose of Window Manager data structures except by calling Window Manager functions.
- You should revise your application so that it accesses the data structures defined by the Window Manager only through accessor functions. The use of these accessor functions, to be provided soon, will give your application greater threading flexibility in Mac OS X.
- You are encouraged to adopt the standard Mac OS 8 window definition procedures in your application. Applications that use the standard Mac OS 8 window definition procedures inherit the Mac OS 8 human interface appearance. Applications that use custom window definition procedures work correctly, but because custom definition procedures invoke their own drawing routines, Mac OS 8 can't draw these applications with the current appearance.

The functions, structures, and constants of the Window Manager appear in the following interface file:

`MacWindows.h`

Note
`MacWindows.h` replaces the older file `Windows.h`. ◆

MPW and Related Interfaces

This section lists additional interfaces used with the MPW development environment that are not part of Universal Interfaces 3.1. These interfaces are listed alphabetically.

IMPORTANT
While these interfaces are not part of the Mac OS system software, Apple plans to make them Carbon-compliant. ▲

ANSI C Interfaces

The functions, structures, and constants of the ANSI C interfaces appear in the following interface files:

```
ctype.h         stddef.h
float.h         stdio.h
limits.h        stdlib.h
locale.h        string.h
math.h          stropts.h
setjmp.h        time.h
signal.h        values.h
stdarg.h
```

C++ Interfaces

The functions, structures, and constants of the C++ interfaces appear in the following interface files:

`exception.h`	`mistream.h`
`fstream.h`	`new.h`
`generic.h`	`stdexcept.h`
`iomanip.h`	`stdiostream.h`
`iostream.h`	`strstream.h`
`milocom.h`	`typeinfo.h`

Note
`iostream.h` replaces the older file `streams.h`. ◆

MPW C Interfaces

The functions, structures, and constants of the MPW C interfaces appear in the following interface files:

`fcntl.h`	`SizeTDef.h`
`ioctl.h`	`VaListTDef.h`
`SeekDefs.h`	`WCharTDef.h`

MPW Utilities

The functions, structures, and constants used by MPW Utilities appear in the following interface files:

`CursorCtl.h`	`MacRuntime.h`
`ddrt.h`	`MPWLibsDebug.h`
`DisAsmLookup.h`	`PEFBinaryFormat.h`
`Disassembler.h`	`perf.h`
`Disassembler68K.h`	`PLStringFuncs.h`
`ErrMgr.h`	`RTLib.h`
`IntEnv.h`	`SANE.h`
`intrinsics.h`	`Unmangler.h`
`MC68000Test.h`	`XCOFF.h`

These interfaces are only available through static libraries (that is, you must build them directly into your application or MPW tool).